**创智链智能合约辅助开发系统**

**用户手册**

**北京新创智链科技有限公司**

目录

[1、引言 2](#_Toc30103783)

[1.1编写目的 2](#_Toc30103784)

[1.2概念解释 3](#_Toc30103785)

[2、关于合约辅助开发系统 3](#_Toc30103786)

[2.1下载 3](#_Toc30103787)

[2.2 安装 3](#_Toc30103788)

[2.3使用插件，生成框架 4](#_Toc30103789)

[2.4一键生成收据 7](#_Toc30103790)

[2.5自动检查 7](#_Toc30103791)

# 1、引言

## 1.1编写目的

本文档提供的对象主要是合约的开发者用户用户。本文档对合约辅助开发系统的操作进行介绍，为用户操作、使用智能合约辅助开发系统软件提供参考和指导。

## 1.2概念解释

什么是智能合约开发辅助系统。

智能合约开发辅助系统是基于Kotlin 开发的用于goland IDE的一款支持智能合约可行性分析、合约的安全性分析以及合约的公平性分析功能为一体的合约辅助开发工具。

# 2、关于合约辅助开发系统

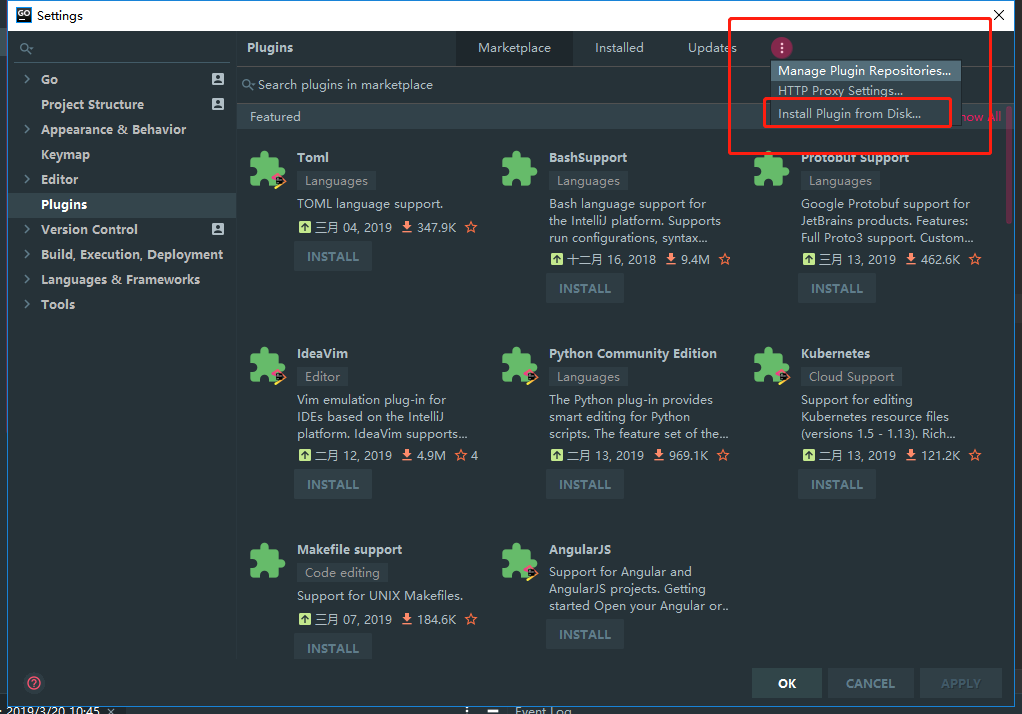
## 2.1下载

本系统的下载地址：

[https://github.com/gichain/sdk/tree/master/plugin](https://github.com/bcbchain/sdk/tree/master/plugin)

## 2.2 安装

1. 选择Goland版本，2018.3.5及以上的版本安装。
2. **打开Goland，选择:File->Settings->Plugins->Install Plugin from disk：**

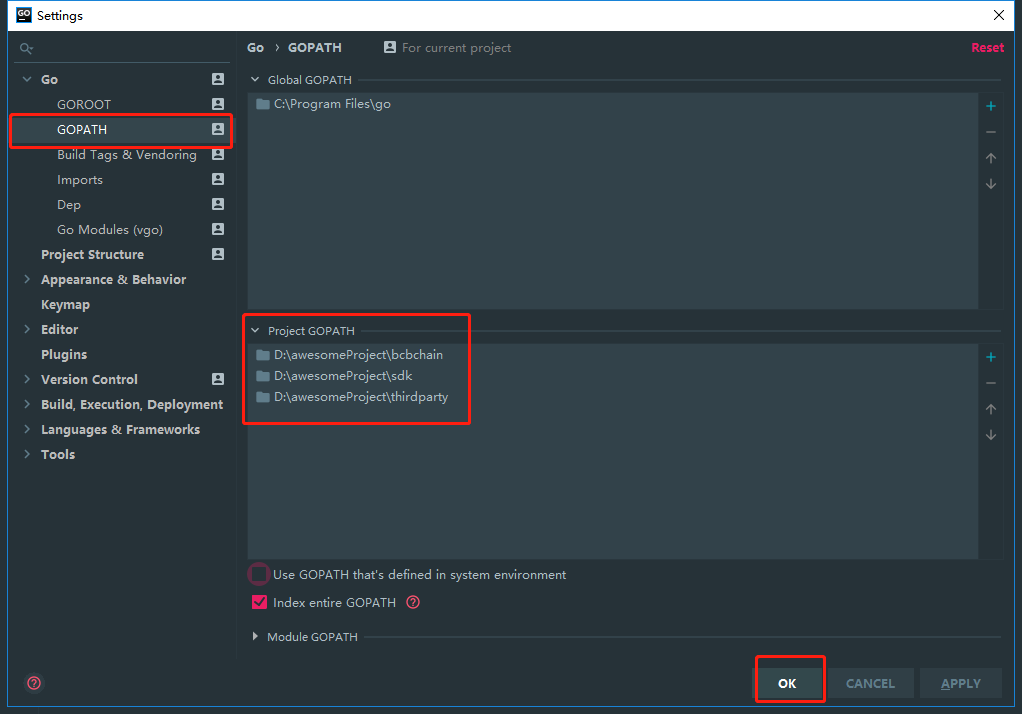


1. **从github上下载的sdk包中找到plugin文件夹找到插件的压缩包进行安装。**
2. **插件安装完成没有提示，直接重启Goland插件。**
3. **安装成功。**

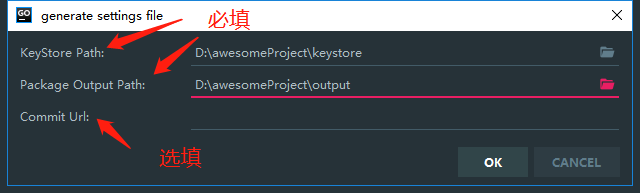
## **2.3使用插件，生成框架**

1. 新建一个工作目录(演示用：d:/awesomeProject，以下简称P)。
2. 从github上下载相关联的包。
3. [https://github.com/gichain/thirdparty](https://github.com/bcbchain/thirdparty)
4. [https://github.com/gichain/sdk](https://github.com/bcbchain/sdk)
5. [https://github.com/gichain/gichain](https://github.com/bcbchain/bcbchain)

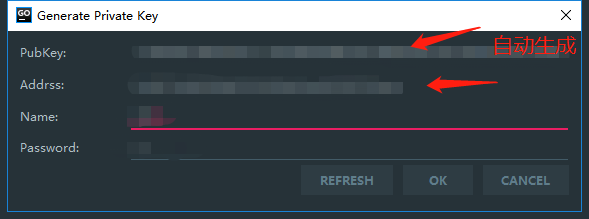
放入同一目录下，在Gopath中添加三种包：



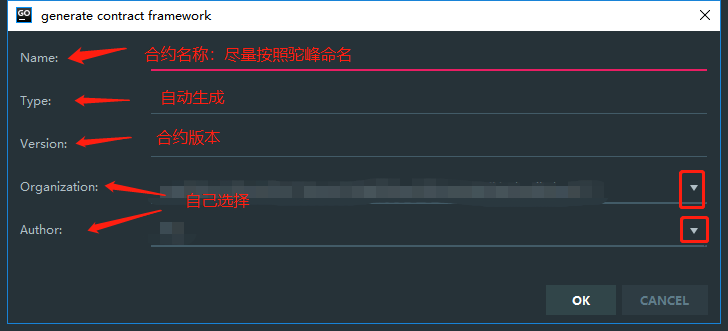
1. 右键选中P->GI Smart Contract->settings
2. 设置keystore 路径和packageoutput路径



1. 设置成功后，右键P ->GI Smart Contract->Private Key,点击GENERATE设置开发私钥，设置成功以后在keystore目录生成对应的文件（两个 \\*.properties和\\*.ks文件）（密码和私钥要牢记备份，否则无法找回）：



1. 上述步骤成功以后，就可以New一个自己的智能合约了，继续右键文件P->New出现下图：根据自己的情况填写



1. 在对应的目录下会生成.go 文件，插件自动生成的代码注释尽量不要修改否则进行合约检查时会失败。

type Dice struct { // 合约中要在数据库中存储的结构体

sdk sdk.ISmartContract

//This is a sample field which is to store in db

//@:public:store

sampleStore string // 在结构体中定义的变量会自动生成对数据库操作的代码。

}

//InitChain Constructor of this Dice

//@:constructor

func (d \*Dice) InitChain() {

//初始化操作，此函数可以没有任何操作

}

//SampleMethod This is a sample method

//@:public:method:gas[500] // 合约的示例接口，对外接口需要加上此格式注释，gas值可变

func (d \*Dice) SampleMethod() {

}

1. 在简易的合约代码生成以后可以添加相对应代码，在结构体中添加一个变量，右键菜单使用 Generate code功能，自动生成数据库相关代码：

type Dice struct {

sdk sdk.ISmartContract

//This is a sample field which is to store in db

//@:public:store

sampleStore string

//@:public:store

tokenName map[string]bn.Number //自定义变量 生成代码见下图

}

1. 添加一个合约对外接口，使用 Generate code，会自动生成相对应的测试接口

//@:public:method:gas[500] 注释格式可以对比 自动生成的 SampleMethod()函数

func (d \*Dice) GetAmount(tokenName string) {

if tokenName == "" {

tokenName = d.sdk.Helper().GenesisHelper().Token().Name()

}

sdk.Require(

d.\_chkTokenName(tokenName), // 检查tokenName是否存在

types.ErrInvalidParameter,

"The contract has not support this token",

)

d.\_tokenName(tokenName)

//todo...

}

1. 此时一个简单的合约框就完成了，Generate Code可多次使用，并且在使用时会自动检查合约是否符合开发规范，如果不符合会提示错误。

## 2.4一键生成收据

1. 插件提供一键生成收据功能，收据会返回到链上方便查询等一系列操作，关键字 receipt ，在合约中添加此结构体并在其中定义一个接口，依旧利用Generate Code 功能 。

//@:public:receipt //必须有此注释

type receipt interface { // 固定格式 前缀必须为emit

emitTokenAmount(tokenName string, amount bn.Number)

}

1. 将生成的收据函数添加到刚才的合约函数GetAmount中，这样一个完整的简易对外接口就完成了。

//@:public:method:gas[500]

func (d \*Dice) GetAmount(tokenName string) {

if tokenName == "" {

tokenName = d.sdk.Helper().GenesisHelper().Token().Name()

}

sdk.Require(

d.\_chkTokenName(tokenName),

types.ErrInvalidParameter,

"The contract has not support this token",

)

amount := d.\_tokenName(tokenName)

//todo...

d.TokenAmount(tokenName, amount) // 收据函数

}

## 2.5自动检查

1. 插件提供自动检查功能，在GI链的合约开发规范中，会禁止某些库的调用。可以使用 插件中的 Check功能来进行检查：

比如 map类型的数据结构禁用了 range功能，而需要用合约提供的mapx.ForRange()下面是代码示例

//@:public:method:gas[500]

func (d \*Dice) GetAmount(tokenName string) {

if tokenName == "" {

tokenName = d.sdk.Helper().GenesisHelper().Token().Name()

}

sdk.Require(

d.\_chkTokenName(tokenName),

types.ErrInvalidParameter,

"The contract has not support this token",

)

amount := d.\_tokenName(tokenName)

//todo...

tmp := map[string]bn.Number{"1": bn.N(1000), "4": bn.N(3000), "2": bn.N(2000)}

for k, value := range tmp {

// 此种range的用法被禁止，利用Check功能检查会报错会报错

d.\_setTokenName(k, value)

}

d.TokenAmount(tokenName, amount)

}

1. 下面为正确的用法

//@:public:method:gas[500]

func (d \*Dice) GetAmount(tokenName string) {

if tokenName == "" {

tokenName = d.sdk.Helper().GenesisHelper().Token().Name()

}

sdk.Require(

d.\_chkTokenName(tokenName),

types.ErrInvalidParameter,

"The contract has not support this token",

)

amount := d.\_tokenName(tokenName)

//todo...

tmp := map[string]bn.Number{"1": bn.N(1000), "4": bn.N(3000), "2": bn.N(2000)}

//for k, value := range tmp {

// // 此种range的用法被禁止，利用Check功能检查会报错会报错

// d.\_setTokenName(k, value)

//}

mapx.ForRange(tmp, func(k string, value bn.Number) {

// 正确的用法

d.\_setTokenName(k, value)

})

d.TokenAmount(tokenName, amount)

}

1. 关于插件的相关功能简单介绍结束。